

**CSCI321 – Project (Android Packet Sniffer)**

**Project Proposal**

**28th April 2018**

Group: SS18/2A

Soh Yu Xuan

Timothy Chin

Kenneth Huang

Kendrick Tan

Supervisor: Mr Tian Sion Hui

Assessor: Mr Tan Kheng Teck

**Document Control**

Title: Project Proposal

Document Name: SS182APP.docx

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Owner | Current Version | Last Change on | | Approved by |
| Date | Time |
| Soh Yu Xuan | 0.7 | 28th April | 2223 |  |

**Distribution List**

|  |  |
| --- | --- |
| Name | Title/Role |
| Soh Yu Xuan | Team Leader / Lead Programmer |
| Timothy Chin | Assistant Programmer / Lead UI |
| Kenneth Huang | Assistant Programmer / Lead Tester |
| Kendrick Tan | Project Management |

**Record of Revision**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Revision Date | Description | Section Affected | Changes Made By | Version after Revision |
| 21th April | Filled Executive Summary & Introduction | Executive Summary & Introduction | Soh Yu Xuan | 0.1 |
| 23rd April | Filled Research | Research | Timothy Chin | 0.2 |
| 24th April | Completed Research & Objectives | Research & Objectives | Soh Yu Xuan | 0.3 |
| 26th April | Completed Scope and Limitations | Scope & Problems of Limitation | Kenneth Huang | 0.4 |
| 28th April | Filled Development Method | Development Method | Kendrick Tan | 0.5 |
| 28th April | Updated Research | Research | Soh Yu Xuan | 0.6 |
| 28th April | Completed Timetable | Timetable | Soh Yu Xuan, Timothy Chin, Kenneth Huang | 0.7 |

**Table of Contents**

|  |  |
| --- | --- |
| Document Control | 2 |
| Executive Summary | 4 |
| Introduction | 4 |
| Project Description | 4 |
| Background | 4 |
| Research | 5 |
| Objectives | 13 |
| Scope and Problems of Limitation | 13 |
| Development Method | 14 |
| Timetable | 15 |
| References | 16 |
|  |  |

**Executive Summary**

Technology has improved with leaps and bounds, computers used to be large clunky machines which were largely immobile but now, almost everyone if not all has a “computer” with them all the time, a smartphone. One of the jobs of a network administrator requires him/her to monitor the network. This can be done with a computer with either with an external network card or with an internal one, however the computer was something that was needed, adding weight and hassle. What if one could monitor and sniff packets from their device whom they carried with them everywhere? One could simply be in a location, capturing packets in their vicinity for analysis, without having to lug around a laptop right into their mobile device.

**Introduction**

The scope of the project has its limitations. An android device differs from a personal computer, in a great many ways. Applications such as wireshark or the aircrack suite on a Linux PC are easy to operate and have been around for decades. Such applications, which capture packets, require a Network Interface Card set in Monitor mode. With a large number of drivers easily downloadable and installable on a personal computer, compatibility and configurability between the NIC cards and the OS make it easy and straightforward as compared to on an android device. Android devices themselves differ in a great many ways. The chipsets from different manufacturers are different, resulting in a lack of universal features. Drivers do not work the same as with a personal computer, for an android device, the native drivers are precompiled in the Linux kernel, making compatibility with external WLAN adapters limited at best without a reverse engineer of existing drivers, integration with the kernel and re-flashing to the new OS. These factors, added to the fact that manufacturers do not/are not supporting development with their hardware make this project very limited and experimental. This document will describe in detail the objectives, scope and limitations of the project.

**Project** **Description**

**Background**

The idea of having airodump-ng albeit a limited version of it on a phone thrills us. Even though android devices are largely segregated by their manufacturers, and the result from this project will be fairly limited, there might be a way to implement such an application on a wider scale with enough time. As with airodump-ng, the user can only capture packets in their vicinity within the range of the NIC, and what better way than to use an android phone, something that weighs so little and are most likely on person most times.

**Research**

**What is packet sniffing?**

“A sniffer (packet sniffer) is a tool that intercepts data flowing in a network. If computers are connected to a local area network that is not filtered or switched, the traffic can be broadcast to all computers contained in the same segment. This doesn’t generally occur, since computers are generally told to ignore all the comings and goings of traffic from other computers. However, in the case of the sniffer, all traffic is hared when the sniffer software commands the **Network Interface Card (NIC)** to stop ignoring the traffic. The NIC is put into promiscuous mode, and it reads communications between computers within a particular segment. This allows the sniffer to seize everything that is flowing in the network, which can lead to the unauthorized access of sensitive data. A packet sniffer can take the form of either a hardware or software solution. [1]

**The Network Interface Controller / Card (NIC)**

To connect to a network, a computer can use a wired connection via Ethernet or wirelessly, which requires a NIC. It can be built into the motherboard or plugged in via USB with an external WLAN adapter. An android device with OTG capabilities (to act as host) with the relevant drivers will be able to detect and use an external WLAN adapter.

**Promiscuous and Monitor Mode**

Monitor mode (RFMON) enables a wireless NIC to capture packets without associating with an access point or ad-hoc network. This is desirable in that you can choose to “monitor” a specific channel, and you never need to transmit any packets. In fact, transmitting is sometimes not possible while in monitor mode (driver dependent). Another aspect of monitor mode it that the NIC does not care whether the CRC values are correct for the packets captures in monitor mode, so some packets that you see may be in fact corrupted. Monitor mode only applies to wireless networks, while promiscuous mode can be used on both wired and wireless networks. Monitor mode is 1 of 6 modes that 802.11 wireless cards can operate in: Master (acting as an AP), Managed (client mode), Ad- hoc, Mesh, Repeater and Monitor mode.

Promiscuous mode allows you to view all wireless packets on a network to which you have to be associated. The need to associate means that you must have some means of authenticating yourself with an access point. In promiscuous mode, you will not see packets until you have associated. Not all wireless drivers support promiscuous mode.

One important difference is Promiscuous mode does tell the card to process all frames (removing 801.11 frame headers) [2]

**IEEE 802.11**

IEEE 802.11 is a set of media access control (MAC) and physical layer (PHY) specifications for implementing wireless local area network (WLAN) computer communication in the 900 MHz and 2.4, 3.6, 5 and 60 GHz frequency bands. They are the most widely used wireless computer networking standards, used in most home and office networks to allow laptops, printers and smartphones to talk to each other and access the Internet without connecting wires. They are created and maintained by the Institute of Electrical and Electronics Engineers (IEEE) LAM/MAN Standards Committee (IEEE 802). The base version of the standard was released in 1997 and has had subsequent amendments. The standard and amendments provide the basis for wireless network products using the Wi-Fi brand. While each amendment is officially revoked when it is incorporated in the latest version of the standard, the corporate world tends to market to the revisions because they concisely denote capabilities of their products. As a result, in the marketplace, each revision tends to become its own standard.

**Layer 2 – Datagrams**

**Management frames**

* Authentication frame
* Association request frame
* Association response frame
* Beacon frame
* Deauthentication frame
* Disassociation frame
* Probe request frame
* Probe response frame
* Reassociation request frame
* Reassociation response frame

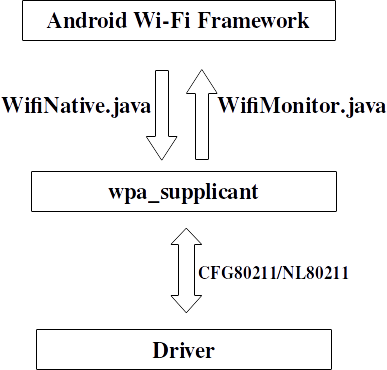
**Control frames**

* Acknowledgement frame (ACK)
* Request to Send frame (RTS)
* Clear to Send frame (CTS)

**Data frames**

Data frames carry packets from web pages, files, etc. within the body. The body begins with an IEEE 802.2 header, with the Destination Service Access Point (DSAP) specifying the protocol, followed by a Subnetwork Access Protocol (SNAP) header if the DSAP is hex AA, with the organizationally unique identifier (OUI) and protocol ID (PID) fields specifying the protocol. If the OUI is all zeroes, the protocol ID field is an EtherType value. Almost all 802.11 data frames use 802.2 and SNAP headers and most use an OUI of 00:00:00 and an EtherType value. [3]

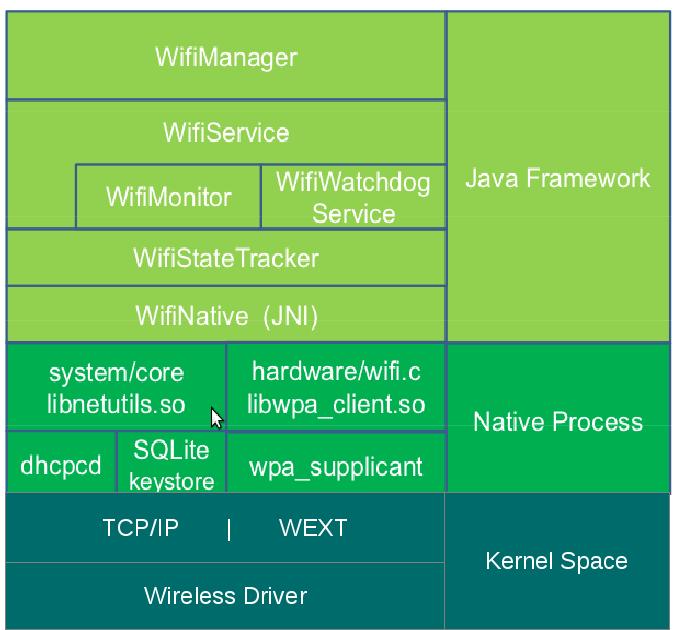
**How Android Wifi State Machine works**



Android uses a customized wpa\_supplicant to perform AP authentication and association, and also communicate with the underlying driver. The **WifiNative** class is used to send various commands to wpa\_supplicant and the **WifiMonitor** class is used to monitor wpa\_supplicant status change and notify the Android framework.

Wpa\_supplicant communicates with the underlying driver using new CFG80211/NL80211 interface [4]

**Android Wifi Architecture**

****

**[5]**

**Root Access on Android**

Root access is not available on a normal android device. Like the sudo command in linux, root access is required to run certain processes in Android. Rooting basically means to give yourself root permissions. However, manufacturers do not support this function at all and by doing so will void the warranty of the device. Rooting of a model of device depends on the community working on it, trying to escalate the default privileges to root. As such, some devices might not have root or exploits available at all.

**PCAP Library – API for packet capturing**

The Packet Capture library provides a high-level interface to packet capture systems. All packets on the network, even those destined for other hosts, are accessible through this mechanism. It also supports saving captured packets to a “savefile” and reading packets from a “savefile”. The pcap API is written in C, with wrapper libraries (Libtins – C++, jNetPcap - Java) available.

**Bcmon/Nexmon – Enabling monitor mode on an android device**

Android devices have their firmware locked down, development of applications which require special privileges are unsupported. Users have to enable root permissions with third party applications which are often experimental, made by enthusiasts who try to reverse-engineer into the device. Bcmon, succeeded by Nexmon is an open project that requires root, it enables devices with select Broadcomm/Cypress WiFi chipsets to enable monitor mode.

**Enabling monitor mode for a specific device (LG G3 – BCM4339)**

1) The first method we explored to tries to achieve the project objective using rooting of phone and modification of drivers

* The LG G3 phone uses BCM 4339 chipset wifi card.
* In order to use the chipset in monitor mode, requires a few things:
  + Rooting the phone
  + Configuration of the drivers/kernel

**ROOTING METHOD 1**

Use either Stump or PurpleDrake to root.

Rooting the phone (Stump Android App)

* ­Enable downloads from “unknown sources” on the phone
* Download the android app called “Stump”
  + <http://downloads.codefi.re/jcase/Stump-v1.1.1.apk>
* Open up the app and hit the “Grind” button
* Next just reboot the phone to get root privileges

Rooting the phone (Purple Drake rooting script)

* This method of rooting requires a PC windows or mac, installation is about the same
* Install PurpleDrake on a windows PC
  + <http://downloads.codefi.re/thecubed/lg_g3/purpledrake/PurpleDrake-TMO_R03.tar.gz>
* Extract out all the files into a folder
* Run the files for the appropriate operating system
  + Windows - purpledrake\_windows  
    Mac - purpledrake\_osx.command
* A command terminal will open.
* Plug in your device and hit **Enter**.
* The device should be rooted.

**ROOTING METHOD 2**

Prerequisites:  
1. [Download LG Root](http://d-h.st/uO8h) ( <http://d-h.st/uO8h> )  
2. [Download LG Drivers](http://androidxda.com/download-lg-usb-drivers) ( <http://androidxda.com/download-lg-usb-drivers> )  
3. A micro USB cable connected to your PC  
4. Your device must be unencrypted

* Rooting through use of PC windows with phone device connected
* Download and unzip the LG\_ROOT.zip file in your desired directory.
* Enable developer debugging on your device by going to “about device” -> “software information” tapping on “build number” until “you are now a developer”.
* In the LG\_ROOT folder, right click inside the folder and click “Open command window here”. Now we have a command prompt running in the current directory.
* At this point, make sure your device is connected to via usb, make sure by typing “abd.exe devices”.
* Next, enter this into the command prompt “adb.exe push busybox /data/local/tmp/ && adb.exe push lg\_root.sh /data/local/tmp && adb.exe push UPDATE-SuperSU-v2.46.zip /data/local/tmp”.
* Now we that the files are in the phone, we want the phone to go into “download mode” to install the firmware. The method to achieve the mode varies device by device.
  + Power off device
  + While powered off, hold volume up and insert usb cable connected to PC
  + Wait for the drivers to install
* Firmware should be updating now.
* Now, double click on “ports.bat” to find out which COM port number the device is on. (Close the window when you have your COM port number)
* Keeping device in download mode, in the cmd prompt type “Send\_Command.exe \\. \COM4”. (make sure COM port number is correct)
* Lastly, run the root script by typing “sh /data/local/tmp/lg\_root.sh dummy 1 /data/local/tmp/UPDATE-SuperSU-v2.46.zip /data/local/tmp/busybox”
* The device is now rooted, now get it out of download mode. (pull the battery out if you have to)

**ROOTING METHOD 3**

This is basically Method 2, but simplified.

Rooting the phone (LG one click root)

LG One Click Root 1.3 (Graphic UI) <https://drive.google.com/open?id=0BxHZt3I9BpoZRjMwN3Uzdm1rbkk>

LG One Click Root 1.2 (The old script method - try this if the Graphic UI version didn't work) <https://drive.google.com/open?id=0BxHZt3I9BpoZbVVQc2dhMldlWUU>

LG drivers <https://drive.google.com/open?id=0BxHZt3I9BpoZQ0xWa1pCX1d5NzA>

* Install LG driver if not already installed
* Download LG One Click Root file and extract
* Enable Developer debugging mode on device settings > about phone > software information > tap build number. (tap build number until developer mode appears)
* Connect phone to PC via usb cable
* If using GUI version, just run “LG One Click Root” installer.
* If using script version, just double click the “LG Root Script.bat” file.
* If phone not recognized, switch between MTP & PTP options
* If error “MSVCR100.dll is missing”, install Visual C++ Redistribution x86
  + <https://www.microsoft.com/en-us/download/details.aspx?id=5555>
  + 64 bit users also install this version, not the 64-bit version

**Controlling root privileges (SuperSu Android App)**

* Making sure apps do not gain root without our permission, control the privileges.
* To control the root privileges, install another app “SuperSu”
  + <https://tag.wonderhowto.com/supersu/>
* Open up the app and make sure to select **Normal** when asked while installing.
* Now reboot the phone and there will be root access and control over it.

**Controlling root privileges (Root Explorer App)**

* Go to the app store and download “Root Explorer”
* It is a powerful tool, however it costs “$5.13” SGD to purchase on the app store.

**Modification of drivers/kernel to enabling monitor mode**

* A lot of the code that modifies the drivers uses flashing of other operating systems.
* The phone LG G3 uses the chipset wifi card “BCM4339”.
* Projects like “NexMon” create firmware for various chipsets, including “BCM4339”. However this was designed with nexus phones in mind.
  + <https://github.com/seemoo-lab/nexmon>

Based on the above research, we have some level of understanding of the context, requirements and limitations of our project. There are two ways to implement packet sniffing with an android device, and both have their limitations. The first way is to connect an external WiFi adapter via an OTG cable (with a device that supports OTG). The limitations to this method is that, depending on the wireless card, specific drivers are required. In android, drivers are precompiled in the Linux kernel, and to update to the required driver, the kernel has to be rewritten with the driver, and the OS has to be re-flashed with the new kernel. This process of reverse engineering of the driver and recompiling of the kernel is extremely difficult and time consuming. Another way is to utilize the in-built NIC of the device. This method is limited by the base device, only a tested device that works will be able to run the sniffer. Both methods require root access to the device, escalating root privileges to the user. Rooting a device is unsupported by the manufacturers and therefore, is highly dependent on open source communities and may differ from model to model. Once a device has a NIC in monitor mode, the application which will be developed, with the help of the PCAP library can then capture packets, save it to a file and possibly provide some sort of an analysis of the data captured.

**Objectives**

We plan to achieve these functionalities

* GUI
* Capture packets with filtering options
* Manipulate data captured, display data
* Save all captured packets to a file for analysis on PC

Possible additions of function

* Cracking WEP, requires packet injection
* Driver reverse engineering and implementation on the application itself instead of re-flashing an updated kernel, allows for usage of external NICs

**Scope and Problems with Limitation**

* Device MUST have a chipset that allows NIC in monitor mode
* Device MUST be rooted
* Bcmon/Nexmon to enable monitor mode

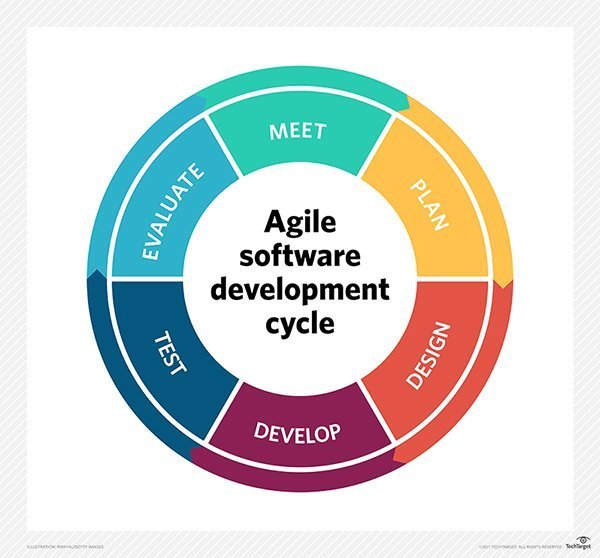
Choice of language

* Java / C++

IDE

* Android Studio

**Development Method – AGILE**



AGILE Software Development Method

* Provides the ability to create and respond to change to succeed in an uncertain and turbulent environment
* We feel that AGILE is the most applicable due to the unpredictability of the project, rooting and enabling monitor mode on specific onboard WLAN cards of chosen devices are experimental and untested.
* There are no true examples on the Google Play Store, true packet sniffing requires root and monitor mode on the onboard NIC, which is unsupported.

AGILE’s Manifesto

* Individuals & interactions > Processes & Tools
* Working Software > Comprehensive Documentation
* Customer Collaboration > Contract Negotiation
* Responding to change > Following a plan

12 Principles of AGILE Software Development Method

* Customer satisfaction by early and continuous delivery of valuable software
* Welcome changing requirements, even in late development
* Working software delivered frequently
* Close, daily cooperation between clients and developers
* Projects are built around motivated individuals, who should be trusted
* Face-to-face conversation is the best form of communication (co-location)
* Working software is the primary measure of progress
* Sustainable development, able to maintain a constant pace
* Continuous attention to technical excellence and good design
* Simplicity – the art of maximizing the amount of work not done is essential
* Best architectures, requirements and designs emerge from self-organizing teams
* Reflect on how to become more effective and adjust accordingly

**Timetable**

|  |  |
| --- | --- |
| 29th April | Overall use case |
| 6th May | 1st Prototype (Simple GUI) + Documentation (Usecase + Class) (Hello World) |
| 13th May | Set up test devices with monitor mode  Understanding PCAP API and relevant libraries |
| 20th May | Understanding PCAP API and relevant libraries  Start of 2nd Prototype |
| 27th May | Continuation of 2nd Prototype |
| 3rd June | End of 2nd Prototype + Documentation (Displaying of raw frames) |
| 10th June | Start of 3rd Prototype |
| 17th June | Continuation of 3rd Prototype |
| 24th June | End of 3rd Prototype (Displaying of packets) + Documentation, Planning of test cases |
| 31st June | Debugging, testing for conformity to test cases |
| 7th July | Working 3rd Prototype |

**References**

1. <https://www.techopedia.com/definition/4113/sniffer>
2. <http://lazysolutions.blogspot.sg/2008/10/difference-promiscuous-vs-monitor-mode.html>
3. https://en.wikipedia.org/wiki/IEEE\_802.11
4. <http://jhshi.me/2014/04/25/how-android-wifi-state-machine-works/index.html#.WuQSE4gRUwo>
5. <https://mitulmodi.wordpress.com/2012/03/21/android-wifi-architecture-wext/>